Improving Reinforcement Learning with Ensembles of Different Learners

Gerrie Crafford[1*] and *Benjamin* Rosman[2]

[1]Council for Scientific and Industrial Research, South Africa
[2]School of Computer Science and Applied Mathematics, University of the Witwatersrand, South Africa

Abstract. Different reinforcement learning (RL) methods exist to address the problem of combining multiple different learners to generate a superior learner. These existing methods usually assume that each learner uses the same algorithm and/or state representation. We propose an ensemble learner that combines a set of base learners and leverages the strengths of the different base learners online. We demonstrate the proposed ensemble learner's ability to combine the strengths of multiple base learners and adapt to changes in base learner performance on various domains, including the Atari Breakout domain.

1 Introduction

In recent years, Reinforcement Learning (RL) has proven its ability to solve a variety of different problems, from superhuman game performance, to solving robotics tasks. Despite RL's success in many areas, a designer is still faced with a choice of a plethora of different algorithms, each with many parameters to tune, in order to design an RL system capable of solving any given problem.

Selecting the best algorithm or parameters for an algorithm beforehand, or even combining the benefits of different approaches is challenging because of the different properties of each algorithm that often only become apparent after the task has been solved and the algorithm's performance has been analysed.

More generally, given a few different learners (which could even have different state representations), some learners might perform better on some tasks while performing worse on other tasks due to the properties of the learner or algorithm. Consider, for example, a model-based learner and a model-free learner training on the same task. The model-based learner would usually have better sample complexity than the model-free learner, but if the model-based learner's model was inaccurate, its performance or sample complexity would degrade, and the model-free learner would instead perform better.

Ensemble machine learning methods are those that use multiple base models and combine the results of the models to obtain improved outputs. Ensemble methods are used successfully in many machine learning applications, including in RL where they can be used to combine multiple policies or value functions to obtain an agent with improved

* Corresponding author: gcrafford@csir.co.za

performance and/or sample complexity. A variety of methods can be used to determine which action the agent should take, from rank voting to Boltzmann multiplication [1] or even using TD-learning to learn weights to use when combining the action-value functions of the set of base agents [2]. Existing ensemble RL methods use a set of learners that use the same internal representations (all learners use action-value functions internally, for example) and the same state representations [3].

We propose a novel algorithm, Adaptive Probabilistic Ensemble Learning (APEL), which is an ensemble learner capable of handling the more general case (handling learners with different state representations and learning mechanisms). APEL runs multiple base learners online and selects the base learners that have the best performance throughout training while only requiring base learners to be able to learn from off-policy experience and making no assumptions about the internal mechanics of base learners. APEL samples learners from a Dirichlet distribution which shifts probability mass to base learners that perform well, while still allowing periodic random exploration of other base learners to allow it to adapt to changes in the performance of the base learners over time.

We show that in tasks where base learners sometimes perform well and other times perform poorly, APEL performs better than any of the individual base learners on average, i.e. APEL's average performance over multiple tasks is better than any of the base learners' average performance over multiple tasks, and outperforms the only other ensemble learning method directly related to APEL.

The contribution of this work is an ensemble agent capable of: (1) outperforming any individual base learner on average over multiple tasks, (2) adapting to changing base learner performance, and (3) leveraging the strengths of multiple base learners online. This agent does not have potential negative societal impacts other than the potential negative impacts of the chosen learners provided to the agent.

2 Background

2.1 Ensemble learner setting

We consider the case where a high-level learner has access to multiple base learners and needs to decide which base learner to use at each timestep. We call this high-level learner an ensemble learner in this work. In this setting, there are two timescales that are of interest: the base learner timescale, t, which denotes a timestep within an episode, and the ensemble learner timescale, e_t, which denotes an episode. Base learners make decisions and learn at each timestep t. The ensemble learner, on the other hand, only makes decisions and learns at the start of each episode e_t (not at each timestep within the episode).

An ensemble learner is similar to bandit algorithms in the sense that it can take actions and receive rewards. It acts by selecting a base learner for an episode and the reward it observes is the episode's return (the cumulative reward obtained by the selected base learner while acting over the entire episode). An ensemble learner can also keep track of certain state information to aid it in choosing base learners and in this case the ensemble learner is similar to a contextual bandit.

2.2 Reinforcement Learning

RL problems are often framed as Markov Decision Processes (MDPs), where an MDP, M, is a tuple (S, A, R, P) [4]. S, and A are the state and action spaces of the MDP. $P(s'|s, a)$ is the probability of transitioning to state s' if action a is taken while the agent is in state s, and $R(s, a, s')$ is the reward r obtained by the agent by taking action a in state s and subsequently transitioning to state s'. A trajectory, τ, of length T is a sequence of (state, action, reward) tuples, $\tau = \langle(s_1, a_1, r_1), \dots, (s_T, a_T, r_T)\rangle$, generated by an agent through interaction with the MDP.

In this work we consider episodic MDPs, where an episode is terminated either when the agent reaches a terminal state, or when the agent has interacted with the MDP T times.

2.3 Different types of learners

In this work, we propose an ensemble learner that remains agnostic to the inner workings of its base learners, where a learner's inner workings are its learning mechanism and state/action representation. Here we give an overview of the different learning mechanisms and state/action representations that learners can use.

The learning mechanisms used by RL learners can be broadly divided into four classes: (1) algorithms that learn (action) value functions in order to determine a policy, such as Q-learning [5] and DQN [6], (2) algorithms that learn policies directly through policy gradients, such as REINFORCE [7], (3) algorithms that combine the two through actor-critic methods, such as Soft Actor-Critic (SAC) [8] and Actor-Critic with Experience Replay (ACER) [9], and (4) algorithms that learn a model of the world and then use that model to learn a policy, e.g. Model-Based Policy Optimization (MBPO) [10]. Learners can use either discrete or continuous state spaces. Their action spaces can also be either discrete (Q-learning, DQN) or continuous (ACER, SAC).

When the learning mechanisms of different learners are the same, information can often be shared between the learners by, for example, averaging over the action-value functions of the learners or by sharing weights between neural networks. When these mechanisms are different, however, it becomes difficult to share information between learners. In the following section we detail our approach to sharing information between base learners while remaining agnostic to their inner workings.

2.4 Learners as black boxes

We take an approach similar to SSBAS [11] and share only trajectories between base learners (instead of sharing action-value functions, etc.). This allows the ensemble learner to consider base learners as black boxes and remain agnostic to the inner workings of the base learners. The ensemble learner is effectively invisible to the base learners since base learners are trained using standard environment interaction in the form of trajectories. Each base learner is then free to use any state representation and method of learning internally.

3 Learning with an Ensemble of Learners

Using an ensemble of RL algorithms or learners can yield improved performance over a single learner as well as better generalisation across tasks [1]. Existing methods, however, often make assumptions about the representations of the learners in the ensemble (requiring all learners to be Q-learners, for example). We consider an ensemble learner that can use a set of base learners while remaining agnostic to the learners' inner workings. The ensemble learner selects different base learners for every episode and then uses the chosen base learner to choose actions to execute during the episode, as depicted in Figure 1. Although the ensemble learner only selects a base learner once per episode, it trains every base learner on each timestep during an episode and is therefore limited to using off-policy algorithms in an episodic setting. Off-policy algorithms are a large class of algorithms and restricting the ensemble learner to the use of off-policy algorithms is therefore not particularly onerous.

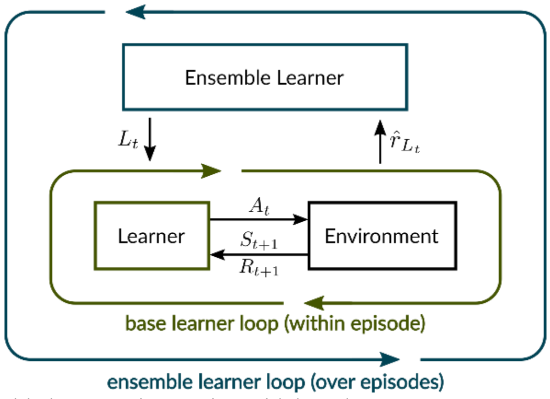


Fig. 1. Diagram of ensemble learner's interaction with base learners.

We define the following as desirable properties for such an ensemble learner: 1. The ensemble learner should select the best performing base learner with high probability. 2. The performance of base learners is non-stationary, and the environment could be non-stationary as well, and therefore the ensemble learner should keep exploring the different base learners throughout training (the probability of selecting the best learner should not become so large that other learners are no longer selected).

We propose the Adaptive Probabilistic Ensemble Learning (APEL) algorithm as an ensemble learner that has the aforementioned properties. The APEL ensemble learner uses a Dirichlet distribution to keep a distribution over the probability of selecting each base learner. The Dirichlet distribution is sampled to obtain a probability distribution according to which a base learner is selected (see Section 3.1 for details). The Dirichlet distribution's probability mass is then shifted to base learners that perform well by increasing their concentration parameters, as described in Section 3.2. Finally, the concentration parameters are decayed each time a base learner is selected (described in Section 3.3). The complete algorithm is described in Algorithm 1.

```
Algorithm 1: Adaptive Probabilistic Ensemble Learning
Require: base learners $L_1, \dots, L_n$, decay parameter $\nu \in [0,1)$
 1: Initialise performance estimates and concentration parameters for each
    base learner.
              $\hat{\boldsymbol{r}} \leftarrow (null, \dots, null), \boldsymbol{\alpha} \leftarrow (1, \dots, 1)$
 2: for each episode $e_t$ do
 3:     Select base learner $L_t$
              $p_{learners} \sim Dir(\alpha), \qquad L_t \sim p_{learners}$
 4:     Generate trajectory $\tau$ with cumulative reward $R$ using base learner $L$
 5:     Train all base learners on trajectory $\tau$
 6:     $\hat{r}_{L_t} \leftarrow R$
 7:     if $\hat{r}_{L_t} > \max(\hat{\boldsymbol{r}} \setminus \hat{r}_{L_t})$ then
 8:         $\alpha_{L_t} \leftarrow \alpha_{L_t} + 1$
 9:     end if
10:     Decay concentration parameters
              $\boldsymbol{\alpha} \leftarrow \max(\boldsymbol{\alpha} \times (1 - \nu), 1)$
11: end for
```

The concentration parameters of the Dirichlet distribution are initialised to 1, i.e., $\boldsymbol{\alpha} = (1, \ldots, 1)$, at the start of training to give all base learners equal probability of being selected. Prior knowledge about the performance of base learners at the start of training can be encoded by assigning different initial values to the concentration parameters corresponding to some of the base learners.

3.1 Learner selection

At the start of each episode et, the ensemble learner selects a base learner L_t by sampling the Dirichlet distribution, $p_{learners} \sim Dir(\boldsymbol{\alpha})$, and then sampling the resulting distribution over base learners to obtain a base learner, $L_t \sim p_{learners}$.

3.2 Adjusting concentration parameters

The ensemble learner keeps track of the performance of each base learner separately, by storing the latest performance estimate for each learner, denoted by the vector $\hat{\boldsymbol{r}}$. At the end of each episode, the performance estimate for the currently selected learner $\hat{r}_{L_t}$ is stored and compared to the stored performance estimates of the other learners. The concentration parameter corresponding to the selected base learner, α_{L_t}, is updated as shown in Equation 1 to increase the concentration if L_t outperformed all the other base learners. Increasing the concentration parameters corresponding to good learners increases the probability of selecting these good learners and allows the ensemble learner to satisfy *property 1.*

$$\alpha_{L_t} = \begin{cases} \alpha_{L_t} + 1, & \hat{r}_{L_t} > \max(\hat{\boldsymbol{r}} \backslash \hat{r}_{L_t}) \\ \alpha_{L_t}, & \hat{r}_{L_t} \leq \max(\hat{\boldsymbol{r}} \backslash \hat{r}_{L_t}) \end{cases} \tag{1}$$

The Dirichlet distribution's concentration parameters are not adjusted until all base learners have been selected at least once, to allow the ensemble learner to have an estimate of the performance of each base learner before it starts to adjust the probability of selecting learners. This is similar to multi-armed bandit algorithms, such as the Upper Confidence Bound (UCB) algorithm [12], where the algorithm is initialised by playing each arm once. In this work we use the cumulative reward obtained by a base learner over an entire episode as the performance of the learner, but other metrics (such as episode length) could also be used.

3.3 Decaying concentration parameters

On tasks where the agent needs to train for a long time to reach optimal performance (most non-trivial tasks), the agent experiences many episodes and this causes the concentration parameters of the Dirichlet distribution to grow without bound, which in turn causes the ensemble learner to become overly confident in a particular base learner (preventing it from still periodically exploring the other base learners).

The large concentration parameters and subsequent overconfidence is curbed by decaying the concentration parameters slightly, by updating the parameters $\alpha \leftarrow \max(\alpha \times (1 - \nu), 1)$, each time after selecting a new base learner (at the start of every episode). The update rule also caps the concentration parameters at a minimum value of 1, to ensure that the probability of selecting any of the underperforming base learners does not become vanishingly small, thereby ensuring that these underperforming learners are still explored periodically.

Decaying the concentration parameters effectively caps the maximum value of the parameters which allows APEL to track non-stationary processes. The concentration parameter decay causes the ensemble learner to have a recency bias, by essentially assigning more weight to more recent observations of base learner performance than to older observations. This probabilistic learner selection mechanism along with the parameter decay allows the ensemble learner to periodically explore the other base learners (which addresses *property 2*), whereas a deterministic approach might settle on one base learner without further exploration of the other base learners.

The concentration decay parameter, ν, can be used to trade off the speed at which the ensemble learner can adapt to changes in base learner performance or environment dynamics with the performance of the ensemble learner. A higher ν allows the ensemble learner to more quickly adapt to changes in base learner performance, while lowering the average performance of the ensemble learner (since suboptimal base learners are selected more frequently).

3.4 Analysis of decay parameters

Assuming, without loss of generality, that there are n learners with one optimal learner L_* whose concentration parameter is incremented at each ensemble learner timestep, then L_*'s concentration parameter at episode k is given by Equation 2 and the steady-state value of the concentration parameter is given by Equation 3.

$$\alpha_k = (1-\nu)^k \cdot (\alpha_0 + 1) + \sum_{i=1}^{k-1} (1-\nu)^i \tag{2}$$

$$\alpha_{ss} = \lim_{k\to\infty} \left((1-\nu)^k \cdot (\alpha_0 + 1) + \sum_{i=1}^{k-1} (1-\nu)^i \right) = \frac{1}{\nu} - 1 \tag{3}$$

Using these equations, the value of ν can be determined according to different desired properties. For instance, an RL practitioner might want suboptimal base learners to be explored with some specified probability or might have a minimum required level of steady-state performance that the ensemble learner must achieve (despite its exploration of suboptimal learners). Given such requirements the decay parameter can be selected according to the desired steady-state exploration probability or steady-state performance by using Equation 4 or 5 respectively. The mean probability of selecting L_* is given by μ_{L_*}, $E[G]$ is the desired steady-state performance and G_{L_N} is the average return of each base learner.

$$\nu = \frac{\mu_{L_*} - 1}{\mu_{L_*} \cdot (1 - \sum_1^{n-1} 1) - 1} \tag{4}$$

$$\nu = \frac{G_{L_*} - E[G]}{G_{L_*} + E[G] \cdot (n-2) - \sum_{i=2}^{n} G_{L_i}} \tag{5}$$

The decay parameter can also be selected according to the desired adaptation time, k, by solving Equation 6 for ν.

$$\left(\frac{1}{\nu} - 1\right) \cdot (1-\nu)^k = (1-\nu)^k \cdot 2 + \sum_{i=1}^{k-1} (1-\nu)^i \tag{6}$$

4 Related work

A concept related to our problem of using a set of learners to improve learning is reusing past policies to improve learning, instead of allowing all the policies to learn while training. Policy reuse methods are typically used in a transfer learning setting and are concerned with determining the best previous policy to use at each point in training to bootstrap training on a new task.

Policy Reuse in Q-learning (PRQL) [13], Context-Aware Policy Reuse (CAPS) [14] and Actor Critic with Teacher Ensembles (AC-Teach) [15] are examples of algorithms that reuse previous policies probabilistically for exploration while training a new policy. An alternative method, Bayesian Policy Reuse (BPR) [16], uses a Bayesian approach to select previous policies that maximise the acquisition of useful information about the current task, while minimising the amount of additional regret accumulated.

Instead of training an ensemble of learners on the full task, the original single-agent problem can instead be divided into different subtasks, with a different learner assigned to learning each subtask. Multi-Advisor Reinforcement Learning (MAd-RL) [17] divides a task by factorising the reward and assigning a learner to each source of reward. Divide-and-conquer RL [18] instead partitions the state space into slices and then trains an ensemble of policies on these different slices, while gradually unifying the different policies into a single policy.

The only work that is directly related to our work and also considers the problem of selecting between different learners to optimise online performance while remaining agnostic to the internal mechanics of its base learners proposes two algorithms: Epochal Stochastic Bandit Algorithm Selection (ESBAS) and Sliding Stochastic Bandit Algorithm Selection (SSBAS) [11]. ESBAS uses a batch setting where base learners are not trained after each environment step, whereas SSBAS can be used in an online setting where base learners are trained after every environment step. Exploration and exploitation of base learners is balanced in SSBAS by using the UCB method. SSBAS uses a sliding window that becomes larger as training progresses to allow it to "forget" old evidence. This limits SSBAS's ability to adapt to changes in base learner performance. We compare our approach (APEL) to SSBAS in our experiments.

5 Experiments

In the experimental evaluation, we evaluate APEL's ability to: (1) outperform any individual base learner on average when run on multiple tasks (Section 5.1), (2) adapt to changing base learner performance (Section 5.2), (3) leverage the strengths of multiple base learners online (Section 5.3), and (4) analyse the effect that many additional base learners have on APEL's performance (Section 5.4).

We use multiple different domains. The standard OpenAI Gym Half Cheetah (v2) and CartPole (v1) environments are used as well as the Gym Atari Breakout environment [19] (along with a custom version of the environment described in Section 5.3). We also use a maze domain, that is a 10×10 perfect maze, with wall positions that are randomised on each run of an experiment as well as random starting and goal locations. The maze domain experiments evaluate train time, e_T, which is the number of episodes it takes an agent to reach near-optimal performance. Concretely, e_T is the episode at which the mean length of the last 50 episodes is below $11.1 \times optimal\, length$. CartPole is considered "solved" when the agent's average reward is greater than 195.0, so for CartPole e_T is the episode at which the mean reward of the last 50 episodes is greater than 195.0.

The hyperparameters for APEL and SSBAS were selected by performing a sweep over values and selecting a value that minimised both the train time and adaptation time. The ranges of the sweeps were $[0.001, 0.3]$ and $[0.01, 10]$ for ν and ξ respectively.

5.1 Outperforming individual learners on average

In this experiment, we show that APEL reduces an RL practitioner's design burden when training RL algorithms on new problems. Without prior knowledge about the different RL algorithms and the problem to be solved, an RL practitioner would not know which algorithm to use on a new problem. This experiment evaluates APEL's ability to select the best algorithm while training when given a set of different base learners and faced with different problems.

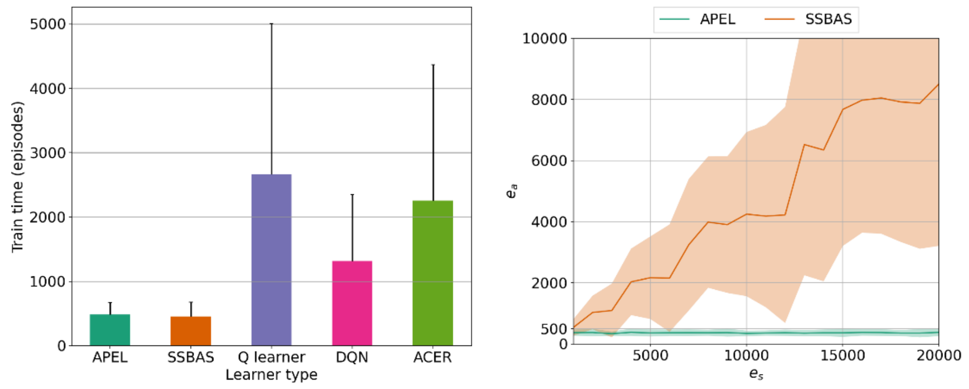
APEL is given three learners as base learners: a tabular Q-learner [5], a DQN learner and an ACER learner. In each run of the experiment, the agent is placed in one of two domains at random: the maze domain or the Cartpole domain. The following hyperparameters are used: Q-learners: learning rate $\alpha = 0.1$, discount factor $\gamma = 1$, $\varepsilon = 0.1$, DQN learner: default hyperparameters used by OpenAI baselines implementation, ACER learner: $\alpha = 2e - 4$, $n\_hidden = 256$, $\gamma = 0$, APEL: $\nu = 0.01$, SSBAS: $\xi = 4.921$. The average training times for the different learners (Q-learner, DQN, ACER and APEL) for both domains are shown in Figure 2a, with the standard deviation shown as error bars. The results are averaged over 100 runs.

Figure 2a shows that APEL outperforms all the base learners, when averaging over all runs from both domains, since different learners perform well on the different tasks. This confirms that APEL can be provided with a set of different types of base learners and it will select the most appropriate learner for a given task while training, allowing it to outperform the base learners on average over different tasks and removing the need for an RL practitioner to select the best algorithm beforehand. SSBAS also outperforms all the base learners, but in the next section we show that APEL is much more adept at handling shifting environment dynamics.

5.2 Adapting to changing learner performance

This experiment evaluates APEL's ability to adapt to changing base learner performance by providing two base learners, one which performs better initially and one that performs better after episode es, at which point the environment dynamics changes. We examine the time it takes APEL to adapt after the dynamics have shifted by using the maze domain with two tabular Q-learners, that use ε-greedy exploration, as base learners and simulating the switching environment dynamics by varying the amount of exploration done by the two learners by changing their exploration parameters, ε, at the switch point, e_s, and keeping them fixed otherwise. The base learners start with $\varepsilon_1 = 0.4$, $\varepsilon_2 = 0.8$ and then at e_s the second learner changes to $\varepsilon_2 = 0.01$. The time (number of episodes) it takes the ensemble learner to adapt (switch to selecting the better learner, except when exploring), $e_a = e_T - e_s$, is analysed. The time the ensemble learner takes to adapt, e_a, is shown for different switch points, e_s, in Figure 2b, with the shaded region showing the standard deviation. The results are averaged over 100 runs.

Figure 2b shows that APEL is able to adapt to changes in base learner performance relatively quickly and adapts to changes quicker than SSBAS on average and with significantly less variance in the time it takes to adapt. Even more significant than APEL's ability to adapt quicker than SSBAS on average, is its ability to adapt to changes in base learner performance in constant time regardless of the switch point. This is in contrast to SSBAS that takes longer to adapt the later the switch comes and shows that APEL is more suitable for complex problems (like Atari or Mujoco domains) where it takes many thousands of episodes of training to converge to near-optimal performance and one of the base learners might only start outperforming the others after many thousands of episodes.



(a) Mean train time for learners, averaged over both domains.

(b) Time to adapt to base learner changes for different switch points.

Fig. 2. Diagram of ensemble learner's interaction with base learners.

5.3 Leveraging the strengths of multiple learners online

In this experiment, we evaluate APEL's ability to optimise online performance by selecting the best base learner throughout training, not just in the limit. APEL's performance is compared to its base learners' performance throughout training to determine if APEL achieves the same performance as the best base learner throughout training. Two different domains are used to evaluate APEL's online performance: Atari Breakout and Mujoco Half Cheetah.

5.3.1 Atari Breakout

We use the Breakout domain with two base learners: a tabular Q-learner and a DQN learner. The DQN learner uses the normal (pixel) representation of Breakout and the Q-learner uses a handcrafted representation to simplify learning. The handcrafted reward is simply $+1$ when the paddle hits the ball (instead of $+1$ when a brick is broken). The handcrafted state space has the following state variables: ball position relative to paddle, ball direction and whether the ball is in the upper or lower half of the screen. The OpenAI baselines [20] DQN implementation is used and the default hyperparameters (for Atari) are used except for the exploration time and exploration fraction, which are $5e6$ and 0.2 respectively. The Q-learner uses the following hyperparameters: $\varepsilon = 0.9$, $\varepsilon_{decay} = 7500, \alpha = 0.05, \gamma = 0.9$. We use $\nu = 0.01$ for APEL and $\xi = 4.921$ for SSBAS. The performance (mean 100 episode reward) of both base learners, APEL and SSBAS is shown throughout training in Figure 3a, with the shaded region showing the standard deviation. The results are averaged over 10 runs.

Importantly, we show here that APEL is able to select between base learners that are completely different, i.e. not just two instantiations of the same algorithm using different hyperparameters, but two different algorithms with different state spaces and learning mechanisms. We also compare APEL's learning curves to SSBAS's learning curves when using the same base learners.

The learning curves show that the handcrafted Q-learner quickly converges to mediocre performance, whereas the DQN learner performs much better but takes much longer to reach that level of performance. APEL's learning curve follows the handcrafted Q-learner for the first part of training (where the handcrafted Q-learner performs better) and follows the DQN for the second part of training (where the DQN performs better), showing that APEL is able to leverage the strengths of multiple base learners online. There is a small delay between the

DQN starting to perform better than the handcrafted Q-learner and APEL selecting the DQN instead of the handcrafted Q-learner. This delay is caused by APEL being relatively certain that the handcrafted Q-learner is the best base learner and having to shift the probability mass to the DQN instead once it explores the DQN and observes that its performance is now higher than the handcrafted Q-learner. The delay is relatively small, however, and confirms Section 5.2's results: APEL is able to adapt to changes in base learner performance relatively quickly even after having selected one base learner for a long time.

SSBAS's performance is similar to APEL's: it also starts out by selecting the handcrafted Q-learner and then later switches to the DQN-learner, but it takes much longer to switch to the DQN-learner once it starts outperforming the handcrafted Q-learner. This delay is consistent with previous results and therefore this lag would, presumably, be greater the later the switch happened.

5.3.2 Mujoco Half Cheetah

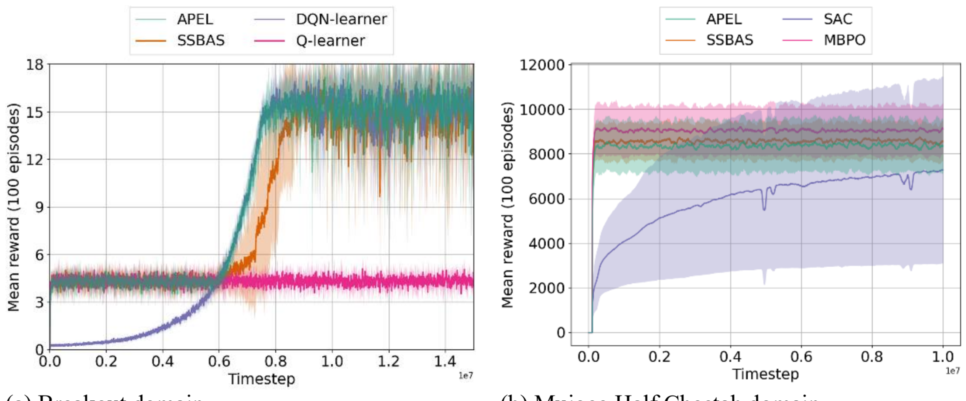
We also run APEL and SSBAS on the Mujoco Half Cheetah domain with a SAC base learner and a MBPO base learner. The authors' implementation of the MBPO algorithm is used with their hyperparameters and the stable baselines [21] implementation is used for the SAC algorithm. MBPO is only allowed to train for 100,000 timesteps after which its training is frozen due to its training being very computationally expensive (compared to SAC). Training is frozen at 100,000 timesteps since the performance does not improve much beyond this point. The learning curves of the learners are shown in Figure 3b with the shaded region showing the standard deviation. The results are averaged over 5 runs.

The results show that APEL and SSBAS correctly select MBPO as the best-performing learner and do not switch to SAC since it never (in the 10 million timesteps) starts outperforming MBPO. This result confirms that APEL is capable of leveraging model-based learners alongside model-free learners and also shows that APEL will select the best base learner online without prior knowledge about the characteristics of each base learner.

5.4 Effect of additional learners on performance

This experiment analyses the effect that adding many redundant base learners has on APEL's performance. We use two configurations for the base learners: many good learners, where the ensemble learner has one bad learner (that is unable to finish the task) and n good learners (that are able to finish the task), and many bad learners, which is the inverse. The experiment is run on the maze domain and the Cartpole domain. The maze domain's "good" learners are the tabular Q-learners and its "bad" learners are ACER learners with small networks (16 hidden units). Conversely, the "good" learners on Cartpole are the ACER learners and the "bad" learners are tabular Q-learners which use a very coarse quantisation of Cartpole's continuous state space, rendering them unable to solve the task. APEL's train time for different values of n (averaged over 100 runs) is shown in Figure 4, with the maze domain's results on the left and the Cartpole domain's results on the right. The standard deviation is shown by the shaded regions.

Figure 4 shows that there is barely any degradation in APEL's performance when many "good" learners are used, and approximately linear degradation in APEL's performance when many "bad" learners are used, which is a small price to pay for the benefits provided by APEL.



(a) Breakout domain. (b) Mujoco Half Cheetah domain.

Fig. 3. Learning curves of different base and ensemble learners on different domains.

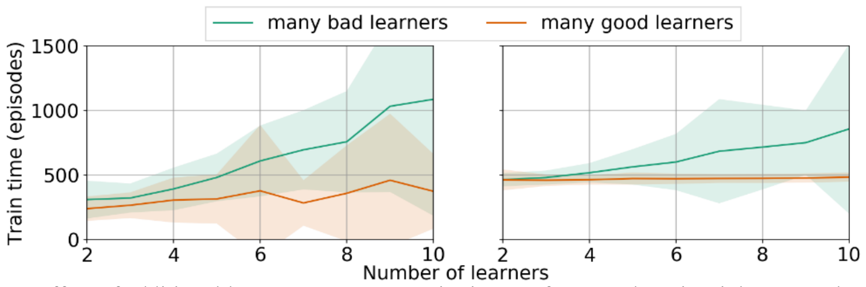


Fig. 4. Effect of additional learner on APEL's train time. Left: maze domain, right: Cartpole.

6 Conclusion

We have presented APEL, an ensemble learner that selects between base learners online by using a Dirichlet distribution to shift probability mass to base learners that perform well. APEL is capable of outperforming individual base learners on average, adapting to changing base learner performance and leveraging the strengths of multiple base learners online. Unlike existing methods in the ensemble RL space, APEL is adept at handling learners that have different internal mechanics.

We showed that, compared to SSBAS, the most similar related method that is also agnostic to the inner workings of its base learners, APEL has similar performance when evaluated on its ability to outperform individual base learners on average, and is able to adapt to changes in base learner performance quicker than SSBAS. Furthermore, APEL's adaptation time stays constant regardless of the number of episodes the agent has experienced, whereas SSBAS's adaptation time scales linearly with the number of episodes the agent has experienced. We also show that APEL is able to leverage the strengths of multiple different types of base learners (from value-based learners to actor-critic and model-based learners) online by evaluating the learning curves of two different base learners and APEL on the Atari Breakout domain and the Mujoco Half Cheetah domain respectively.

References

1. M. Wiering and H. Van Hasselt, “Ensemble Algorithms in Reinforcement Learning,” vol. 38, pp. 930-6, 2008.
2. V. Marivate and M. Littman, “An Ensemble of Linearly Combined Reinforcement-Learning Agents,” in *ConferenceProceedings of the 17th AAAI Conference on Late-Breaking Developments in the Field of Artificial Intelligence*, 2013.
3. R. Y. Chen, S. Sidor, P. Abbeel and J. Schulman, “UCB exploration via Q-ensembles,” p. arXiv:1706.01502, 2017.
4. R. S. Sutton and A. G. Barto, Introduction to Reinforcement Learning, 2nd ed., Cambridge, MA, USA: MIT Press, 2018.
5. C. J. Watkins and P. Dayan, “Q-learning,” vol. 8, no. 3-4, pp. 279-292, 1992.
6. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, “Human-level control through deep reinforcement learning,” vol. 518, no. 7540, pp. 529-533, feb 2015.
7. R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” vol. 8, no. 3-4, pp. 229-256, 1992.
8. T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” in *ConferenceProceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018,* 2018.
9. Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu and N. d. Freitas, “Sample Efficient Actor-Critic with Experience Replay,” in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Conference Proceedings*, 2017.
10. M. Janner, J. Fu, M. Zhang and S. Levine, “When to Trust Your Model: Model-Based Policy Optimization,” in *Advances in Neural Information Processing Systems*, 2019.
11. R. Laroche and R. Feraud, “Reinforcement Learning Algorithm Selection,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Conference Proceedings*, 2018.
12. P. Auer, N. Cesa-Bianchi and P. Fischer, “Finite-Time Analysis of the Multiarmed Bandit Problem,” vol. 47, no. 2–3, pp. 235-256, may 2002.
13. F. Fernández and M. Veloso, “Probabilistic Policy Reuse in a Reinforcement Learning Agent,” in *ConferenceProceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems,* New York, NY, USA, 2006.
14. S. Li, F. Gu, G. Zhu and C. Zhang, “Context-Aware Policy Reuse,” in *Conference Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, Richland, SC, 2019.
15. A. Kurenkov, A. Mandlekar, R. Martin-Martin, S. Savarese and A. Garg, “AC-Teach: A Bayesian Actor-Critic Method for Policy Learning with an Ensemble of Suboptimal Teachers,” in *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, ConferenceProceedings*, 2019.
16. B. Rosman, M. Hawasly and S. Ramamoorthy, “Bayesian policy reuse,” vol. 104, no. 1, pp. 99-127, jul 2016.

17. R. Laroche, M. Fatemi, H. van Seijen and J. Romoff, “Multi-Advisor Reinforcement Learning,” p. arXiv:1704.00756, apr 2017.
18. D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar and S. Levine, “Divide-and-Conquer Reinforcement Learning,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track ConferenceProceedings*, 2018.
19. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, “OpenAI Gym,” p. arXiv:1606.01540, 2016.
20. P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu and P. Zhokhov, OpenAI Baselines, GitHub, 2017. https://github.com/openai/baselines.
21. A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor and Y. Wu, *Stable Baselines,* GitHub, 2018. https://github.com/hill-a/stable-baselines.